RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHRSTUHL
INFORMATIK XI
SOFTWARE FÜR
EINGEBETTETE
SYSTEME

Professor Dr.-Ing. Stefan Kowalewski
Dipl.-Inform. Bastian Schlich

Aachen, 08.November 2004
SWS: V2/Ü2

Systemprogrammierung

WS 2004/2005

Übungsblatt 4

Abgabe der Lösungen: ab 15.11.2004 in den Übungen

Aufgabe 1 (3 Punkte): Wechselseitiger Ausschluss

Eine Lösung des Problems des wechselseitigen Ausschlusses von zwei Prozessen wurde 1966 von L. Hyman vorgestellt. Der Algorithmus stellt den Prozessen P_0 und P_1 folgende gemeinsame Variablen zur Verfügung:

```
bool flag[2];      /* mit FALSE initialisiert
*/
int turn;          /* mit den Werten 0 bis 1, Initialisierung beliebig */
```

Der Prozess P_i arbeitet nach folgender Programmsequenz, wobei P_j den jeweils konkurrierenden Prozess bezeichnet:

```
1  repeat
2       flag[i]:=TRUE;
3       while (turn!=i) do
4       begin
5            while (flag[j]) do
6                 noop;
7            turn:=i;
8       end
9       critical_section(Pi);
10      flag[i]:=FALSE;
11      remainder_section(Pi);
12 until FALSE;
```

Arbeitet das Verfahren korrekt? Prüfen Sie dies anhand der Anforderungen an eine korrekte Lösung des wechselseitigen Ausschlussproblems. Skizzieren Sie für *jede* der Bedingungen den Beweis oder geben Sie ein Gegenbeispiel an!

Aufgabe 2 (2+2=4 Punkte): Atomare Operationen

Eine Variante einfacher, hardwaremäßig nicht teilbarer Operationen sind Decrement-and-Test `dectest(x)` und Test-and-Increment `testinc(x)`. Beide Operationen eignen sich für Variablen, auf die verschiedene nebenläufige Prozesse konkurrierend zugreifen wollen.

```
1  dectest(x)
2       atomar {
3          x--;
4          if(x>0)
5              return 1;
6          else {
7              if (x<0)
8                  return -1;
9              else
10                 return 0;
11         }
12      }
```

```
1  testinc(x)
2      atomar {
3          int result;
   // Hilfsvariable
4          if(x>0)
5              result = 1;
6          else {
7              if(x<0)
8                  result = -1;
9              else
10                 result = 0;
11         }// end else
12         x++;
13         return result;
14     }// end atomar
```

a) Unternehmen Sie einen Lösungsversuch für den wechselseitigen Ausschluss mit Hilfe von `dectest(x)` und `testinc(x)`. Stellen Sie an Beispielen dar, in welchen Situationen Probleme auftreten könnten.

b) Realisieren Sie die Funktionen `signal()` und `wait()` von Semaphoren (mit Warteschlangen) mit Hilfe dieser unteilbaren Operationen.

Aufgabe 3 (1+1+1=3 Punkte): Prozesskommunikation

Zwischen Prozessen gibt es grundsätzlich zwei verschiedene Arten von Kommunikation – die direkte und die indirekte Kommunikation.

Bei der direkten Kommunikation existiert zwischen den Prozessen genau eine Verbindung. Dabei werden folgende Kommunikationsprimitive verwendet:

- `send(P, message)`: sendet eine Nachricht zum Prozess P
- `receive(Q, message)`: empfängt eine Nachricht vom Prozess Q

Die indirekte Kommunikation läuft hingegen über eine Mailbox:

- `send(A, message)`: sendet eine Nachricht zur Mailbox A
- `receive(A, message)`: empfängt eine Nachricht aus der Mailbox A

Die Nachrichten werden in der Mailbox mittels FIFO verwaltet.

In beiden Fällen wird durch die `receive`-Operation der Prozess so lange blockiert, bis er eine Nachricht erhalten hat.

a) Welche Befehlsfolge (`send`, `receive`) muss ein Prozess ausführen, wenn er bei indirekter Kommunikation eine Nachricht von Mailbox A *und* eine Nachricht von Mailbox B empfangen möchte?

b) Wie sieht die Befehlsfolge aus, wenn ein Prozess eine Nachricht von Mailbox A *oder* von Mailbox B (oder von beiden) empfangen möchte?

c) Eine `receive`-Operation veranlasst einen Prozess so lange zu warten, bis mindestens eine Nachricht in der Mailbox vorhanden ist. Manchmal kann es wünschenswert sein, analog dazu eine `send`-Operation erst dann auszuführen, wenn die Mailbox leer ist. Kann man dies unter ausschließlicher Verwendung der beiden genannten Primitive realisieren? Begründen Sie Ihre Aussage.

Aufgabe 4 (7 Punkte): Threads

Viele vorhandene Programme sind für Prozesse mit nur einem eizigen Thread geschrieben. Diese Programme für mehrere Threads umzuwandeln, kann schwieriger sein, als es zuerst den Anschein hat. Überlegen Sie welche Probleme dabei auftreten können. Betrachten Sie dazu unterschiedliche Aspekte der Programmierung, wie zum Beispiel globale Variablen, Verwendung von Bibliotheken, Prozesskommunikation, Programablauf usw. Diskutieren Sie mögliche Lösungen.

Aufgabe 5 (3 Punkte): Semaphore

In der Vorlesung haben Sie das Konzept der Semaphoren gelernt. Die Semaphoren stellen ein Mittel zur Lösung des wechselseitigen Ausschlusses dar. In verteilten Systemen (wie z.B. Workstation-Netze oder Paralellrechner) wird die Synchronisation üblicherweise durch den Austausch von Nachrichten realisiert. Dazu sind folgende Funktionen nötig:

- `send(P, message)`: *nicht blockierendes* Senden einer Nachricht zum Prozess P
- `receive(Q, message)`: *blockierendes* Empfangen einer Nachricht vom Prozess Q

Die Nachrichten in der Mailbox werden mittels FIFO verwaltet.

Zeigen Sie, dass Semaphoren und Nachrichtensysteme äquivallente Konzepte sind.

Hinweis: Der Synchronisationsprozess ist als `sync` ansprechbar. Für die Verwaltung der wartenden Prozesse stehen die folgenden Funktionen zur Verfügung:

- `void enqueue(int pid)`
- `int dequeue(void)`